

Serverless I

CNAE – Cloud Native Architecture & Engineering



Prof. Stefan Tai
Sebastian Werner, Elias Grünewald, Nicola Leschke, Maria Borges

Information Systems Engineering
TU Berlin

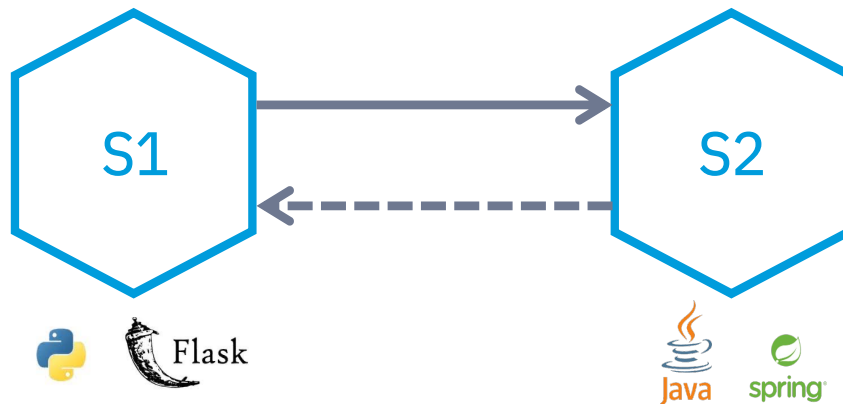
Recap: Challenges in Microservice Applications

Scalability (e.g., too many requests for a single machine)

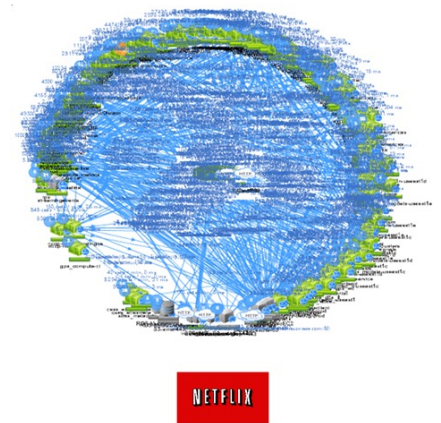
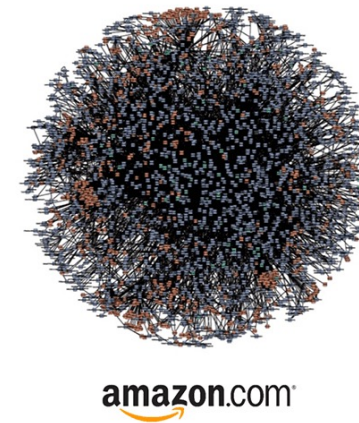
DevOps (e.g., empowering small teams to manage everything)

Observability (e.g., able to observe system behavior)

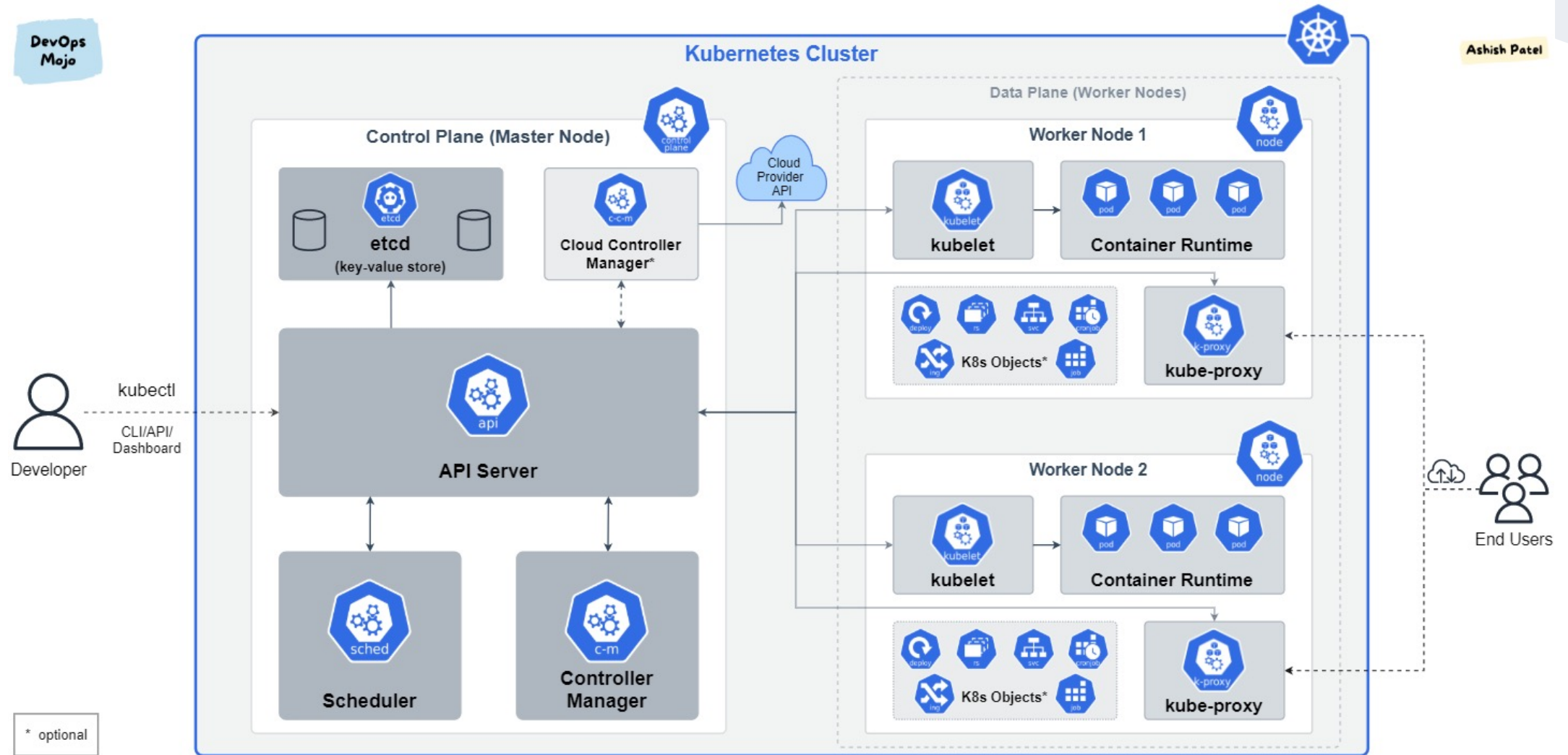
Security ...



VS.



Recap: Kubernetes to the rescue...



Recap: Kubernetes Example - Declarative Operations

my-first-dockerfile

my-first-pod.yml

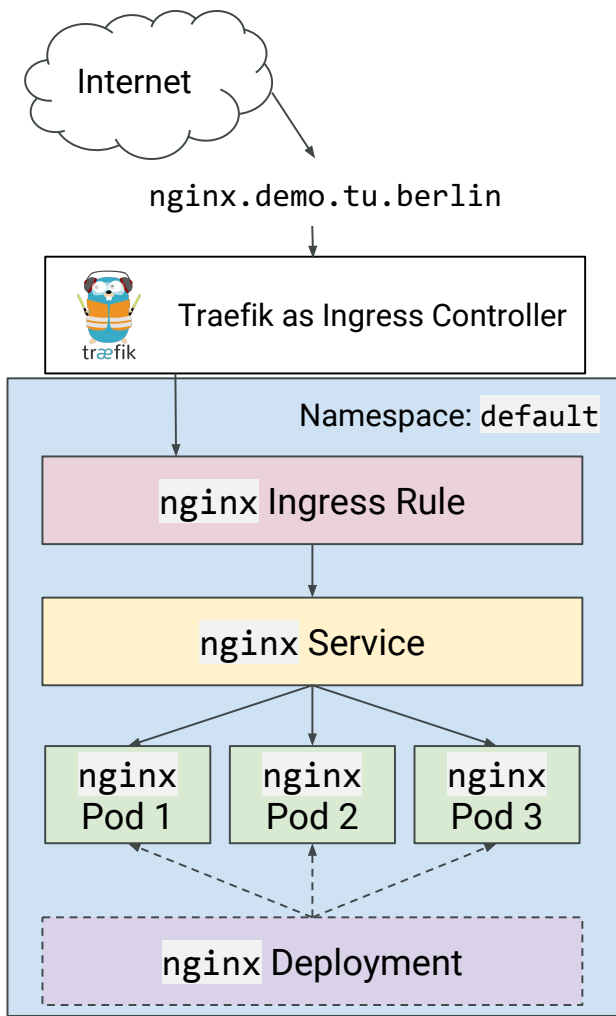
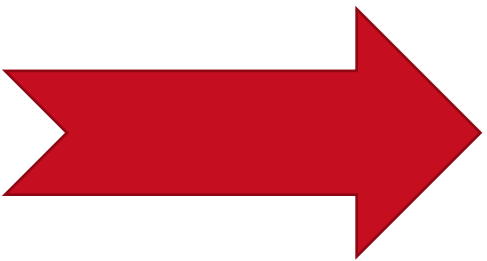
my-first-ingress.yml

my-first-service.yml

```

my-first-pod-and-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
    name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - image: nginx:1.13.9-alpine
        name: nginx

```



Should you use Kubernetes for every microservice application?

Have you used any serverless function before?

Agenda

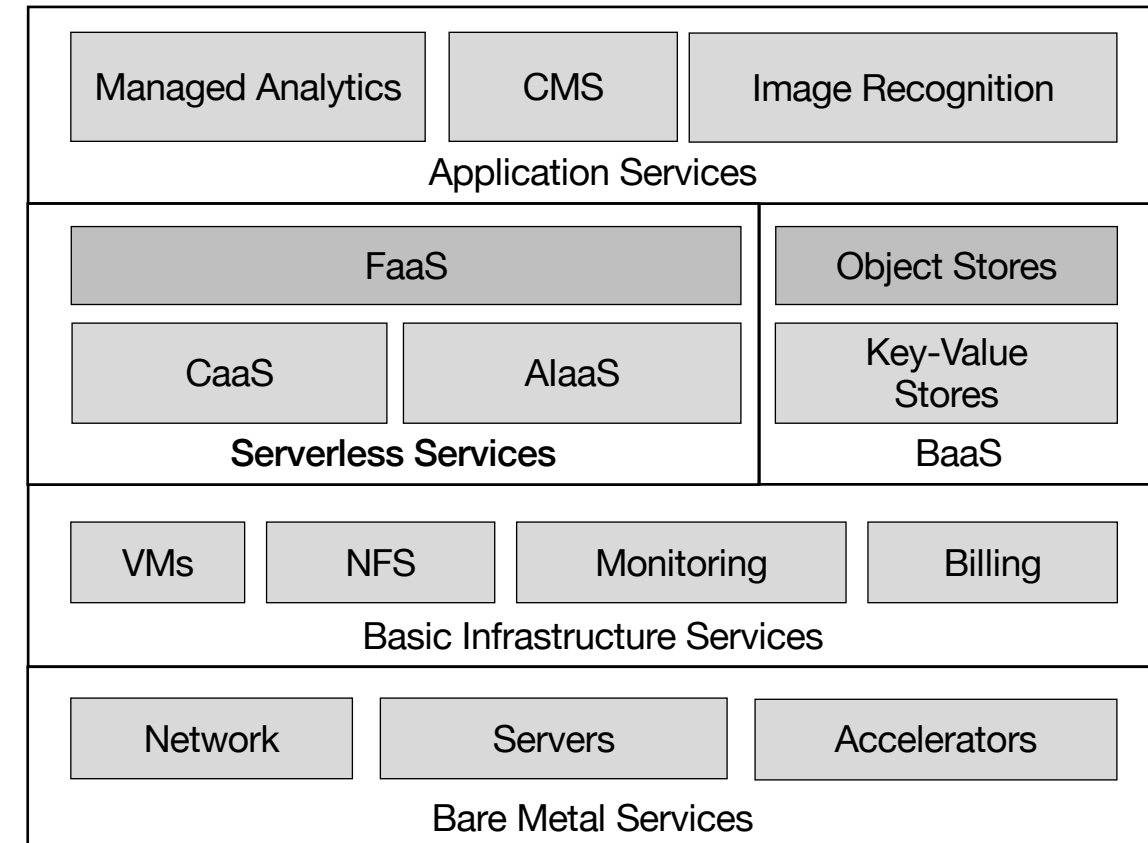
1. What is serverless?
2. Patterns in FaaS-based Applications
3. When and How you can use FaaS
4. Pitfalls and Words of Caution



What is Serverless?

- No clear definition.
- Instead of managing resources, a user provides a task description and the cloud automatically provisions resources to perform that task [1].
- “Server-less models abstract away most of the DevOps related concerns.” [2]
- A serverless service can be controlled with a limited set of configurations that govern application quality and cost [3].
- Typically, glue code/infrastructure between services and applications.
- Very easy to use, start with – hard to get right, hard to maintain.

Serverless Model based on (Jonas et al. 2019)

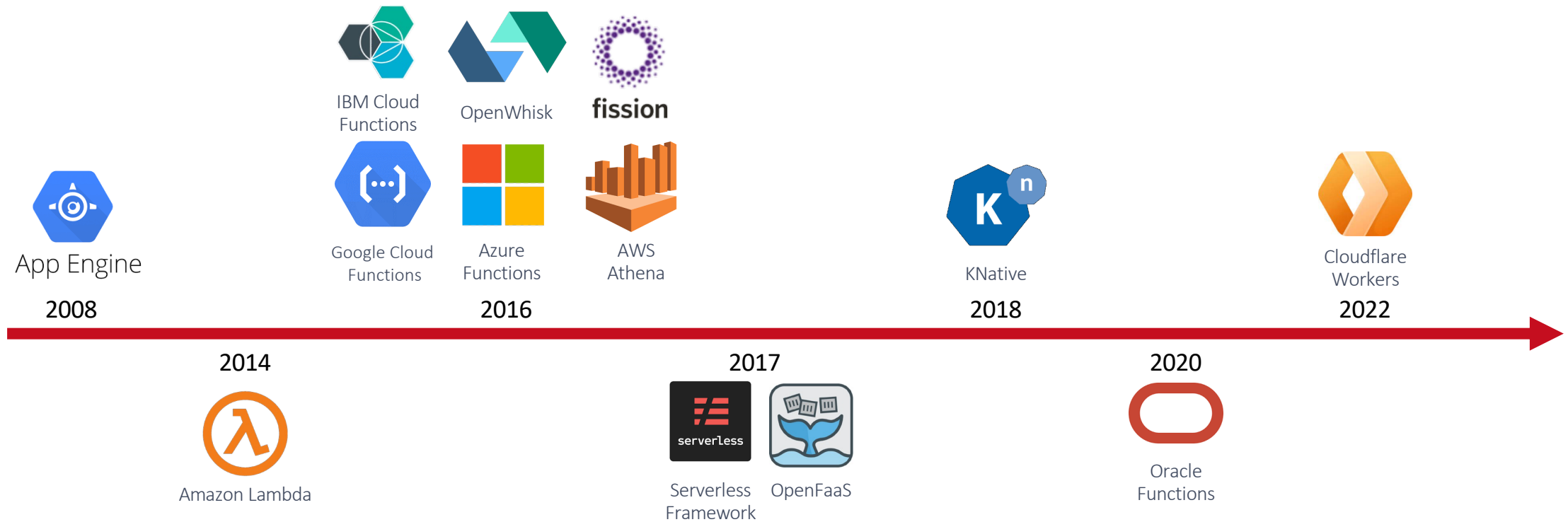


[1] E. Jonas *et al.*, “Cloud Programming Simplified: A Berkeley View on Serverless Computing”

[2] Baldini et al.: Cloud-Native, Event-Based Programming for Mobile Applications

[3] Kuhlenkamp et al.: An Evaluation of FaaS Platforms as a Foundation for Serverless Big Data Processing

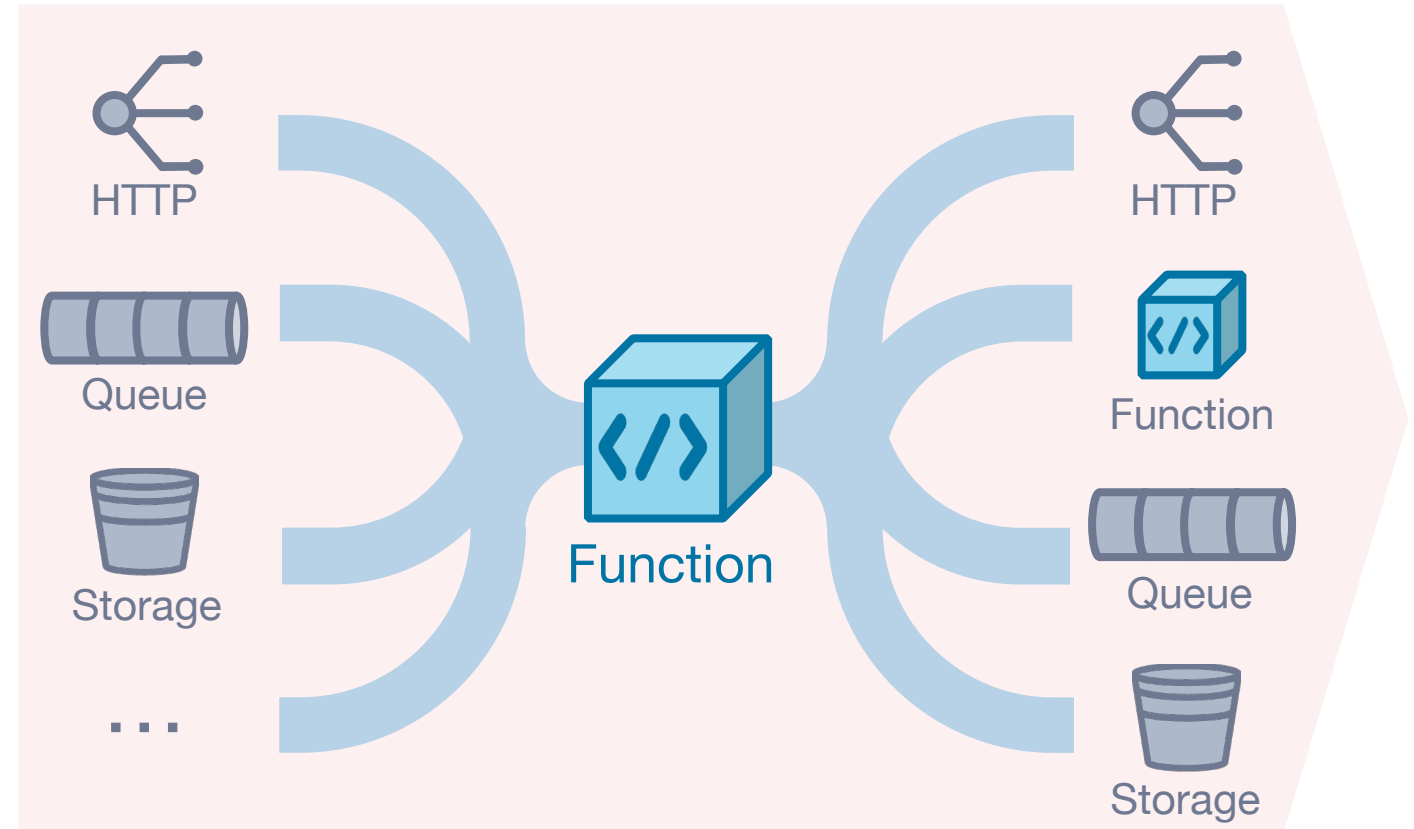
Serverless Timeline



Function-as-a-Service (FaaS)

- **Generic** user-defined managed execution
- execution-based billing model
- Provider-side **managed operational tasks**
- Workload-driven **elastic autoscaling**
- Stateless and ephemeral

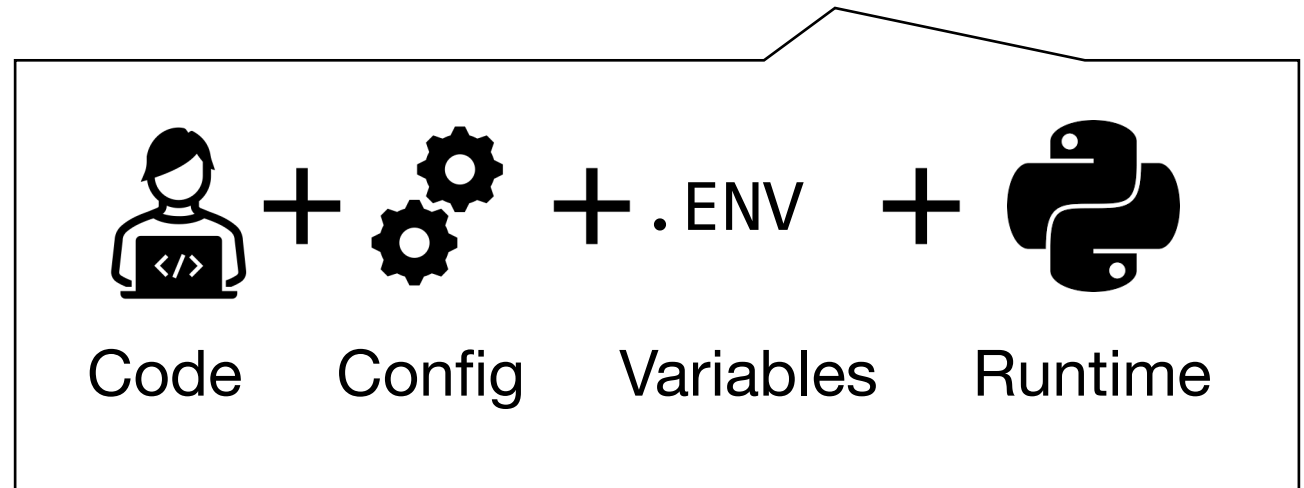
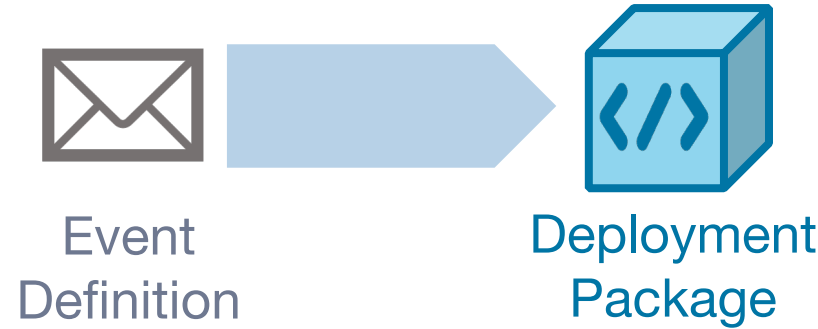
Event-based programming model



FaaS

Programming Model

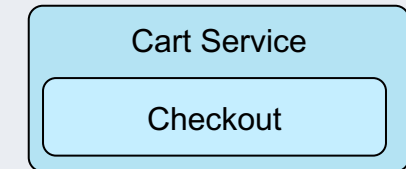
- Select one or more events sources
- Select a runtime environment, e.g., python3.7-arm
- Write the "event-handler".
- Specify configurations, e.g., environment variables or resource limits
- Profit.



What would you use FaaS for?

FaaS Programming Model Example

- check if products in the cart are available
- charge customers' payment method
 - trigger packaging of order
 - update warehouse availability
 - trigger shipment



FaaS Programming Model Example

Source Code

- Many runtimes are supported
- Each specifies a fixed handler method interface
- Deployment packages are restricted in size
- Cold-Start-Problem for large libraries

```
DB = connect()

def handler (event, context):
    for item in event.order:
        if checkAvailability(DB, item.id,item.quantity):
            emitChargeRequest(item,event.account)
            emitPackageOrder(item)
            updateItemQuantity(DB,item.id, -1*item.quantity)
        else:
            error(item,event.order)
    callShipmentFunc(event.order,event.account)
    return dict(status:"ok")
```

package.py

Libraries (must be included by the dev)

Request (3.4kB)

boto3 (1.2kB)

Configurations

- Runtime
- Memory
- Limits
- Trigger (Event-Sources)
- Logging

```
{
  „timeout“:30,
  „memory“:256,
  “concurrent-limit“:1000,
  “runtime“:“python3.7“
}
```

package_config.yaml

Event Sources



HTTP Requests

- using an API-gateway service
- direct call (very bad practice)



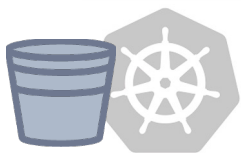
Queue Events

- on insert to topic X



Cron Events

- every 5 minutes



Service Events

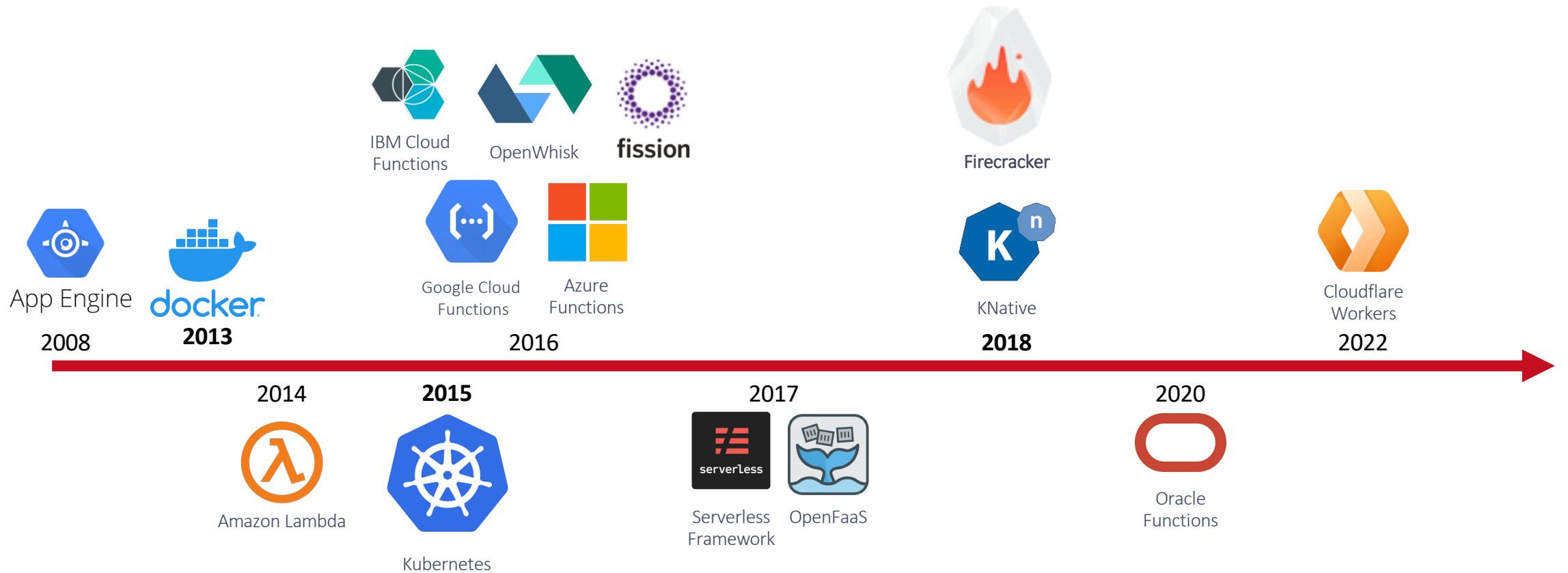
- on new objects
- on config change

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  LambdaFunctionOverHttps:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        HttpPost:
          Type: Api
          Properties:
            Path: '/greetings'
            Method: post
```

Example: SAM-Description¹ for an API-Event for POST /greetings

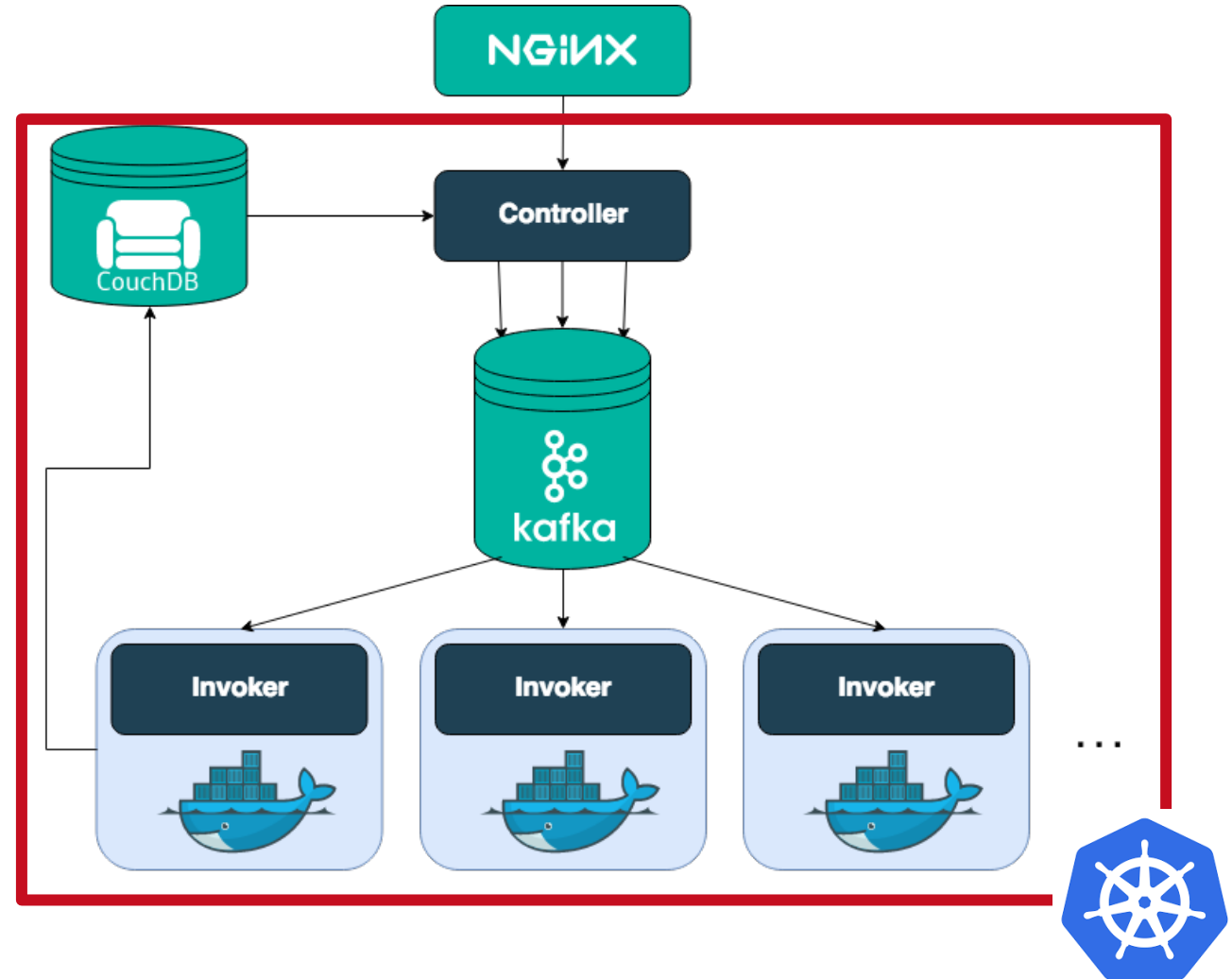


How does this work?



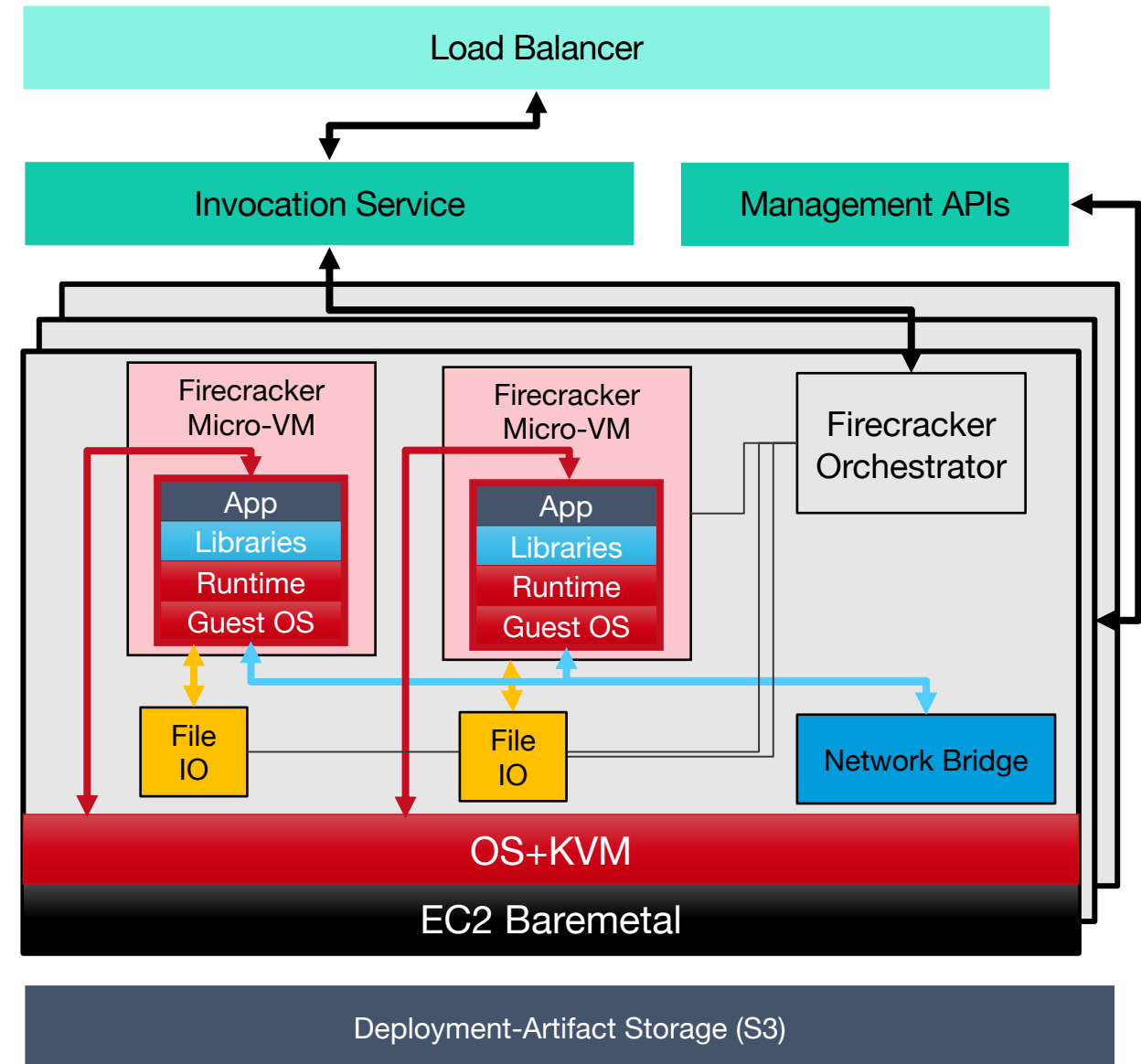
Example: OpenWhisk

- Kubernetes can function as a FaaS-backend
- K8s, manages invoker instances, each invoker can start some function runtimes independently
- FaaS-platform handles invocation distribution, code-storage, result collection



Example: AWS Lambda

- Providers use containers or micro-vms for isolation with predefined runtimes.
- Movement of code/deployment artifacts instead of images (reduced size)
- Runtimes are only provided on a per-event basis and can be created or removed freely by the provider.
- Providers can freely decide how to scale, place executions and when to update → higher resource utilization.



AWS Lambda Architecture

<https://github.com/firecracker-microvm/firecracker>

Example: Simple FaaS-based Application

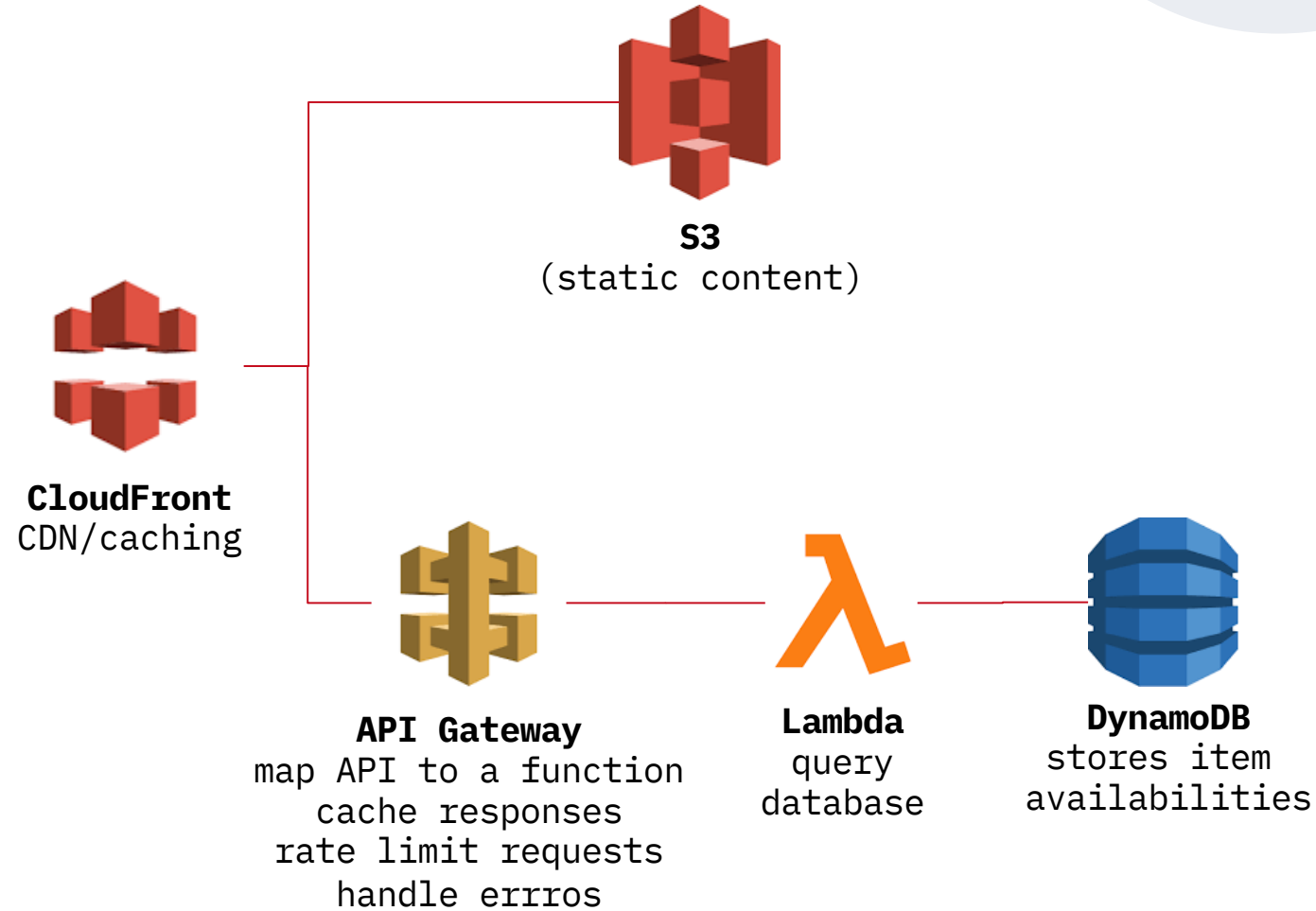
- Generally, a fairly static website, e.g., react-based single page app.
- Some interactive functionality, e.g., product availability check.
- You want an api like:
`/api/v1/products/<id>/variants`



45cm	Schrittlänge: 62 - 68 cm	Ausverkauft ●
48cm	Schrittlänge: 69 - 74 cm	Ausverkauft ●
51cm	Schrittlänge: 75 - 77 cm	Lieferzeit 11 Wochen ●

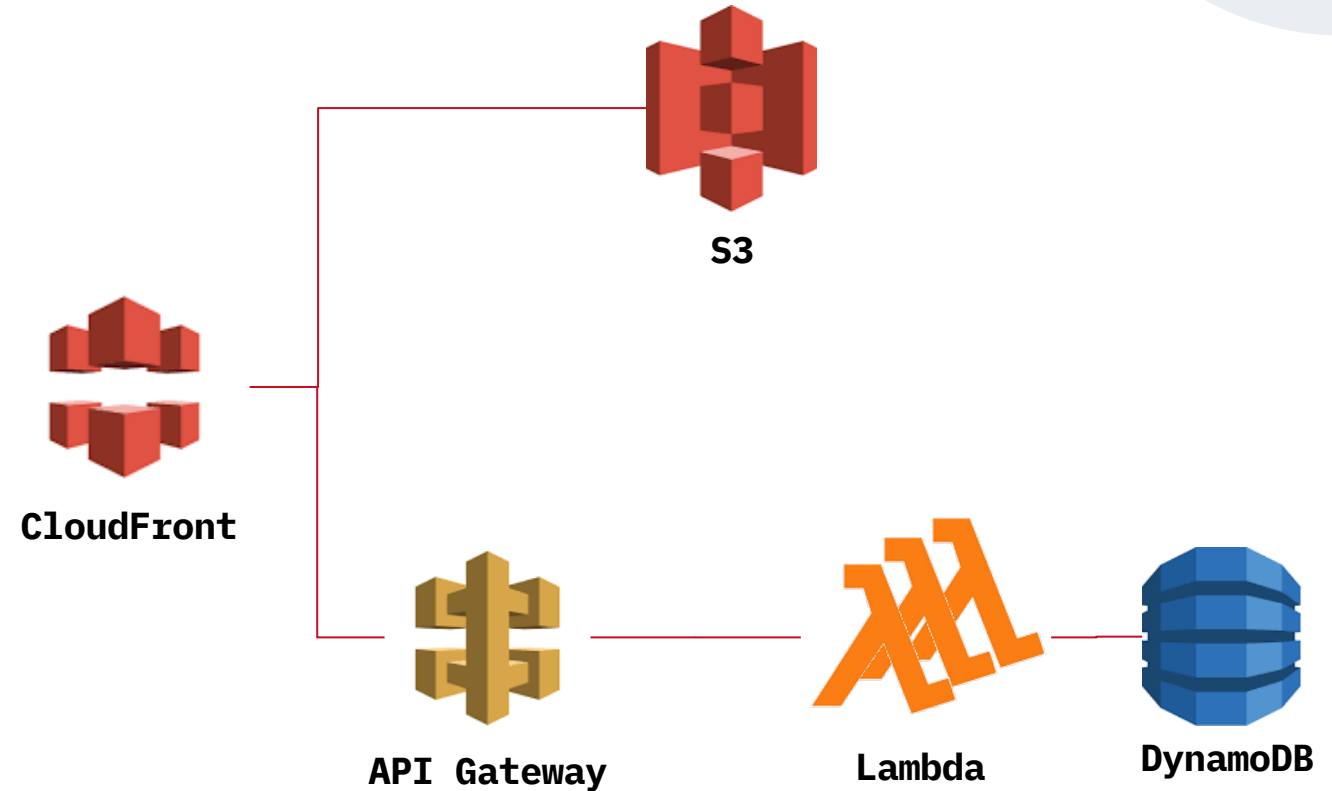
Example: Simple AWS FaaS-based Application

1. Store static content on an object store
2. Use a CDN for quick delivery of results
3. API Gateway to map `/api/v1/products/<id>/variants` to a lambda function
4. DynamoDB to manage the availability data



Example: Simple FaaS-based Application Upside

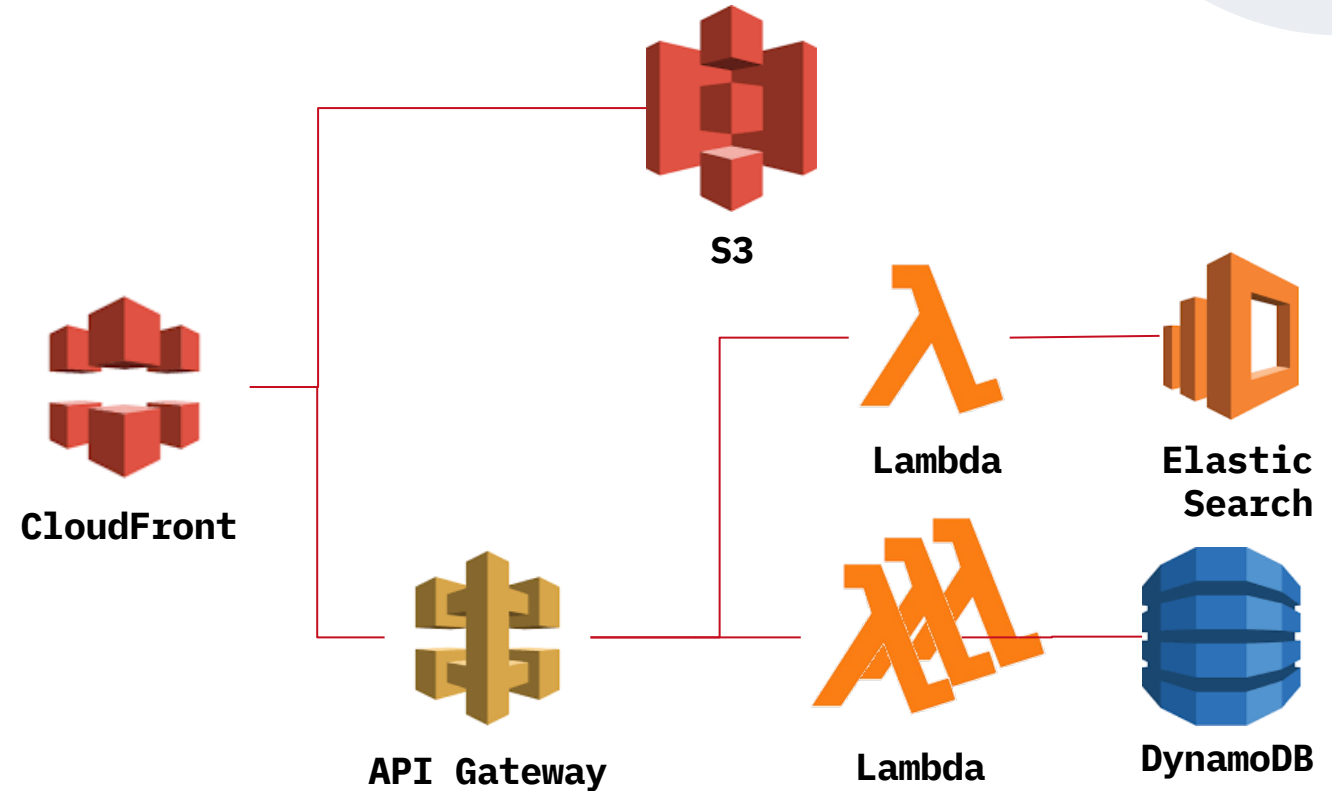
- Can quickly scale to many functions
- Only pay when users use interactive content, e.g., when they are close to buying something
- Function code is very lightweight
→ easy to understand and reason about



Example: Simple FaaS-based Application

Downside

- Increases management responsibilities → a function for every business function/feature
- Uses many services, each with diverging billing and usage models → hard to predict what this will cost
- Highly distributed and complex → hard to find errors



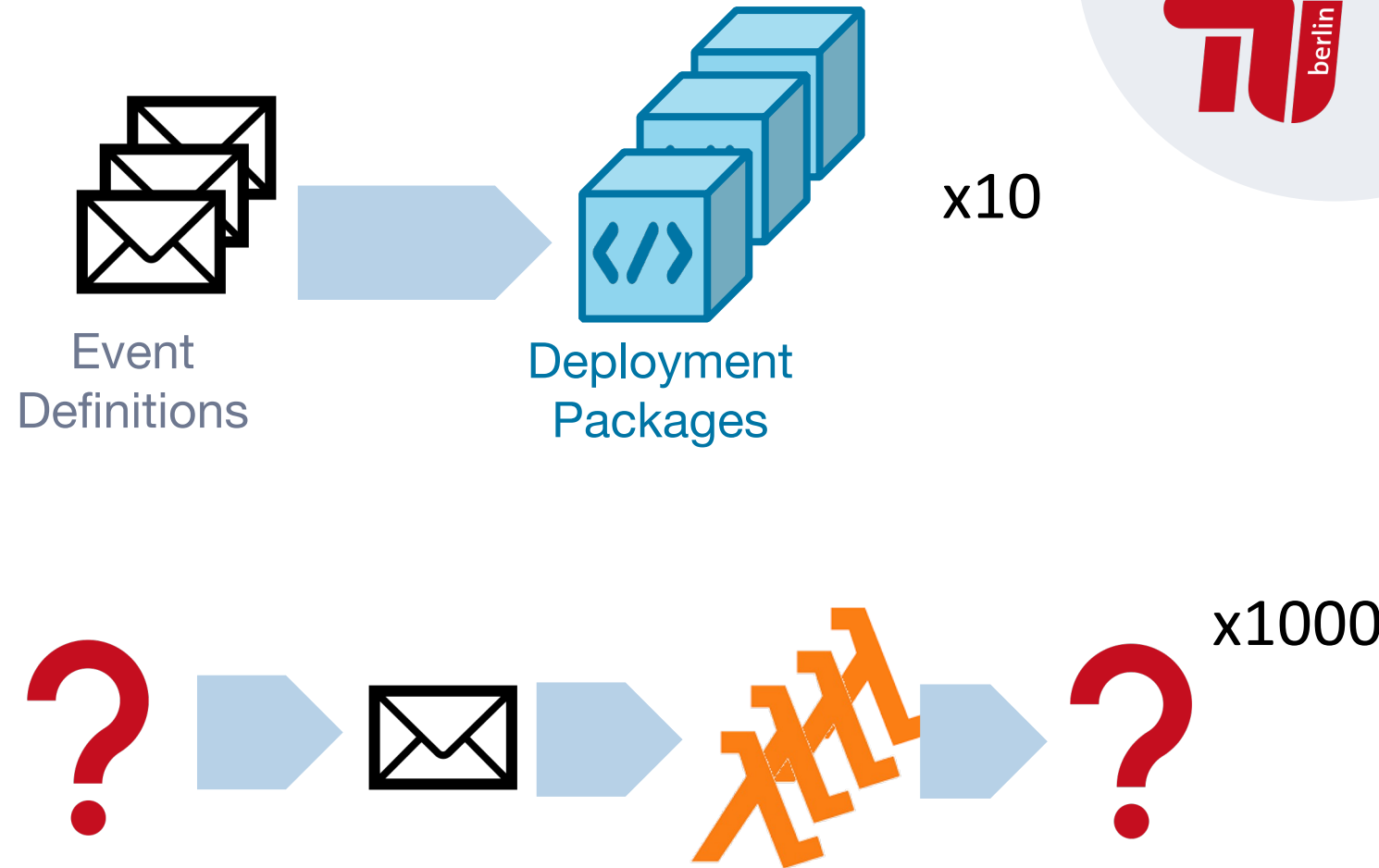
1. What is serverless?
- 2. Patterns in FaaS-based Applications**
3. When and How you can use FaaS
4. Pitfalls and Words of Caution

Always Compositions

- Even simple FaaS-based application are composed
- Composition adds complexity for:
 - Configuration
 - Dev-ops
 - Lock-in
 - Debugging
 - Cost estimation
- Errors in one service might not stop the execution, thus, cost might still occur even for failed or canceled requests, be wary of “denial of wallet” attacks

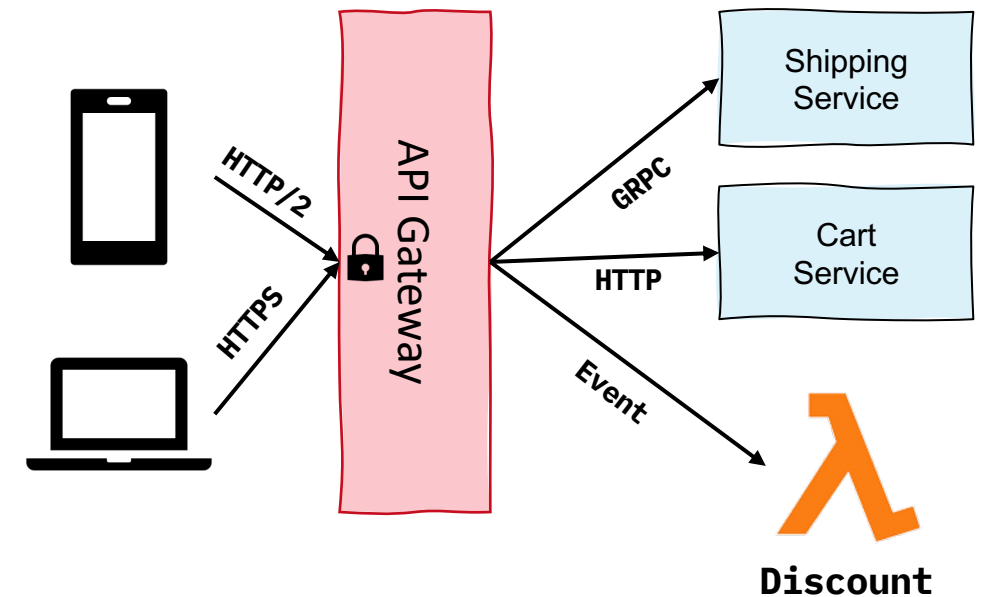
Why is it composed?

- Events $\neq$ HTTP Requests
- Stateless $\rightarrow$ fitting storage
- Lots of deployment packages $\rightarrow$ more shared code
- Lots of executions $\rightarrow$ more “start-up” time



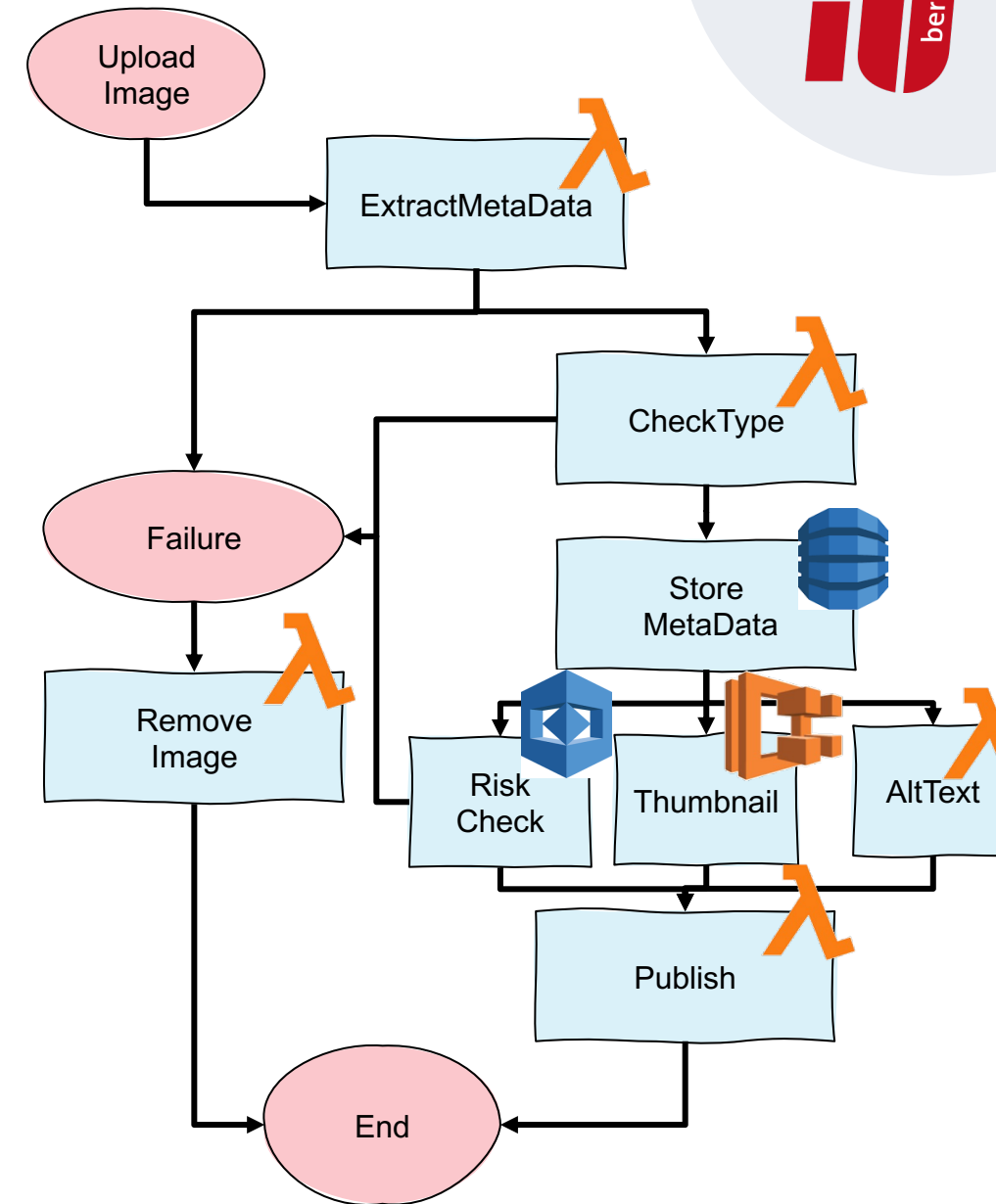
API Gateways

- Not exclusive to FaaS-based applications
- Build in rate limiting, caching, authentication flows
- Can split-up requests (one endpoint, many services)
- Transparent A/B testing, canary releases
- Translate between protocols (e.g., http2, grpc)



Orchestrators

- Allow to manage complex execution flows
- Features may include:
 - branching, conditionals
 - workload-driven parallelism
 - waiting syncing
 - error handling
 - Service combinations
- Can workaround platform limitations

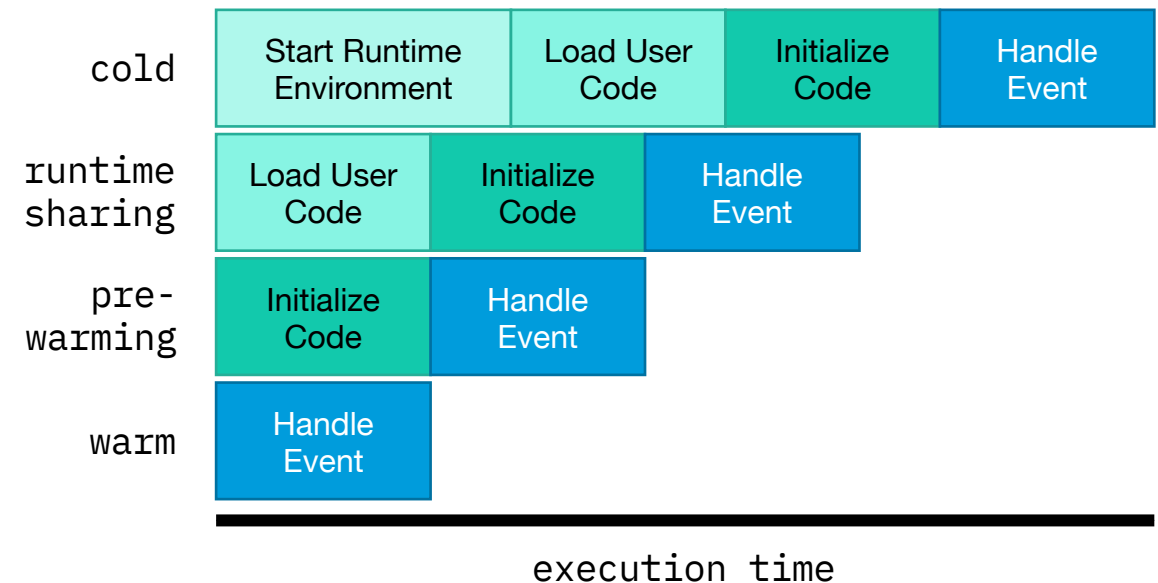


Cold Starts

- Pay for what you use is not free
- Some configuration/platform options:
 - pre-warming (pay for warm instances)
 - runtime sharing (on an account or platform level)

Guidelines:

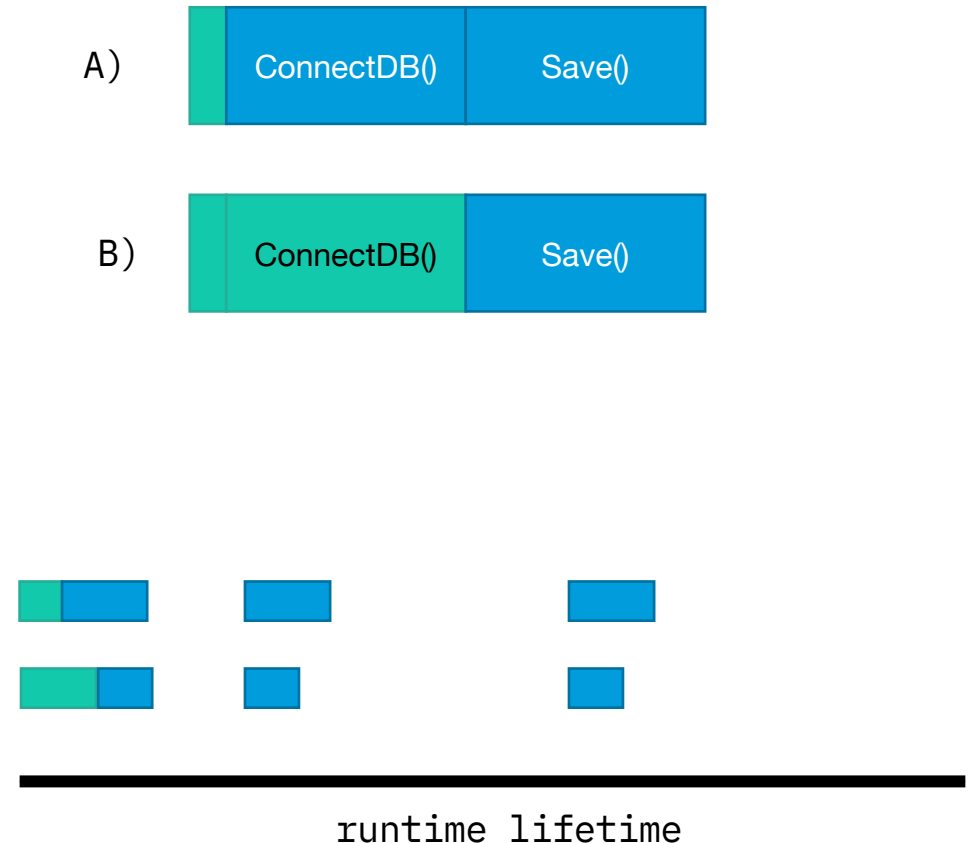
1. Use “fast” runtimes
2. have “small” deployment packages
3. only do the necessary initializations



But

Which option would you implement?

- Both can be valid.
 - Reuse of expensive initialization can reduce follow-up requests -- IF the frequency of requests is within the runtime lifetime window and requests don't come in clusters.
 - Reuse is not sensible IF there is a time-dependent variable, e.g., connection timeouts or frequency or parallelism don't fit



Layers and Packaging

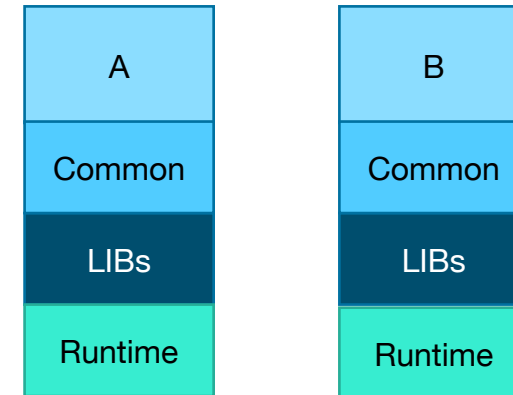
Lots of deployment packages → lots of ops

How to manage libraries and shared code?

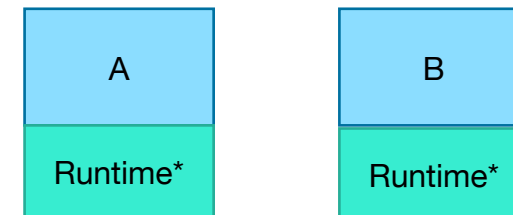
Options:

1. Don't – simple but lots of redundancies and copies
2. Custom runtimes or Layers – less likely prewarmed, makes sense at scale
3. Allround Functions – large deployment packages may increase cold-start times

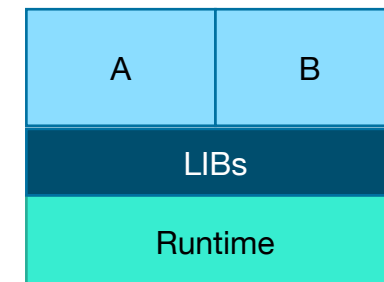
Disjuncted
Deployment
Packages



Custom
Runtimes/
Layers



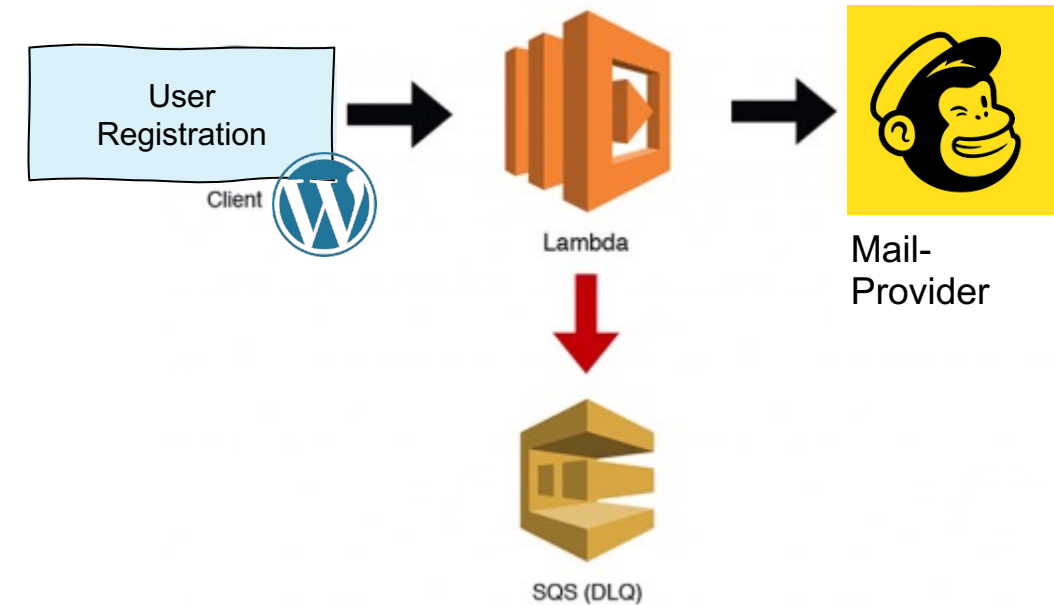
Allround
Functions



1. What is serverless?
2. Patterns in FaaS-based Applications
- 3. When and How you can use FaaS**
4. Pitfalls and Words of Caution

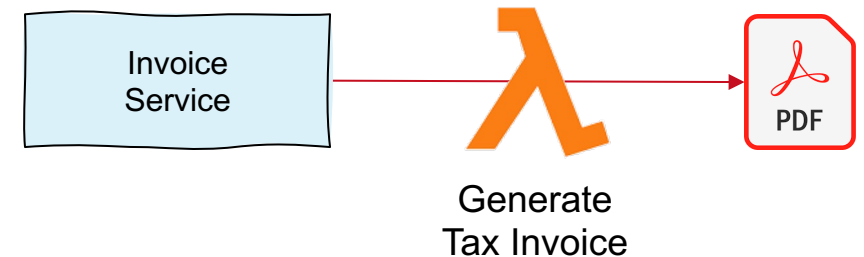
Scalable Glue-Code

- combine managed services “glue them together”
 - translate data models
 - handle/hide access token
- implement policies or failures
 - rate-limiting when using a third-party service to avoid extra cost
 - handle service outages
 - multiplex events, e.g., store email and subscribe to Ad-service...



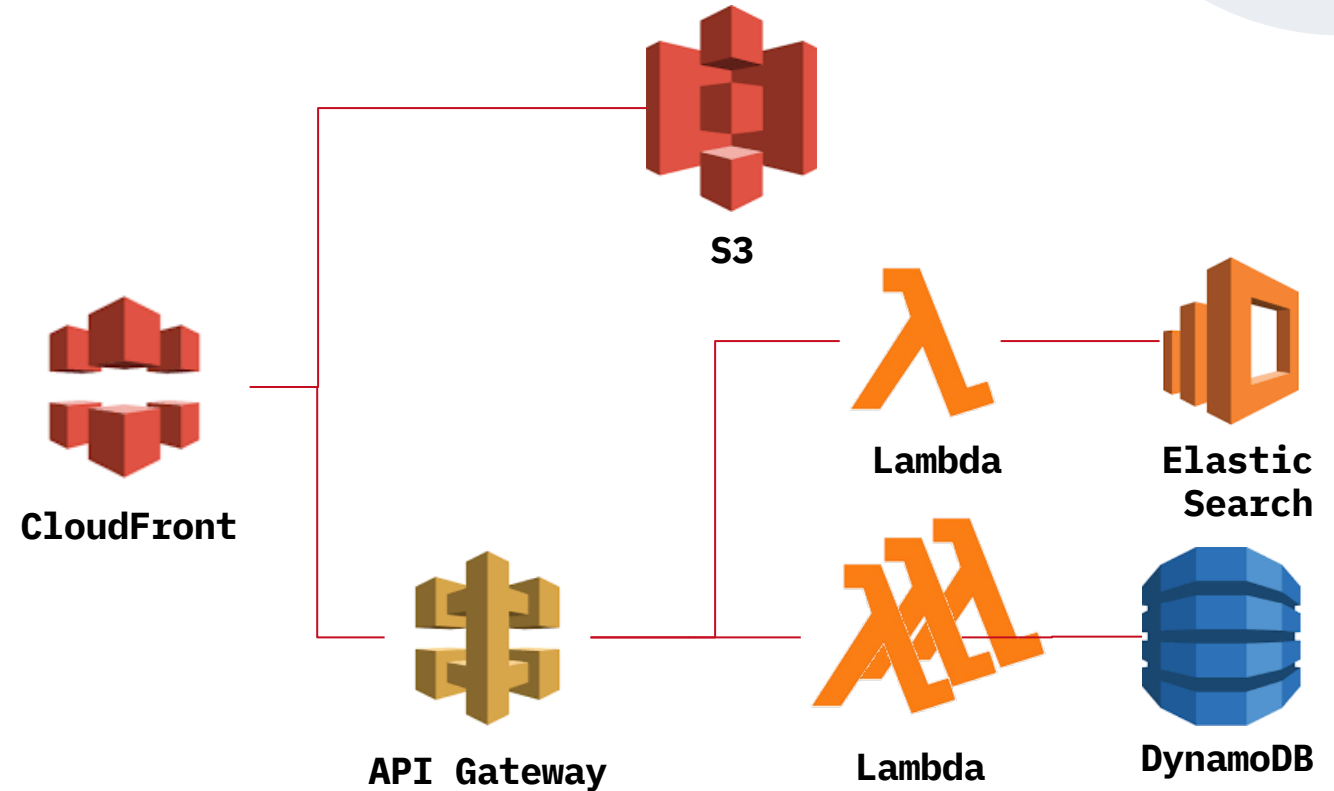
Infrequent Tasks

- some functionality might require significant resources but only is used infrequently
- we don't want to provision resources for "just in case" requests
- simplify core service logic (reduce deployment dependencies)
- suited of async tasks, e.g., the result will be sent via email in a few seconds instead of returned on page visit



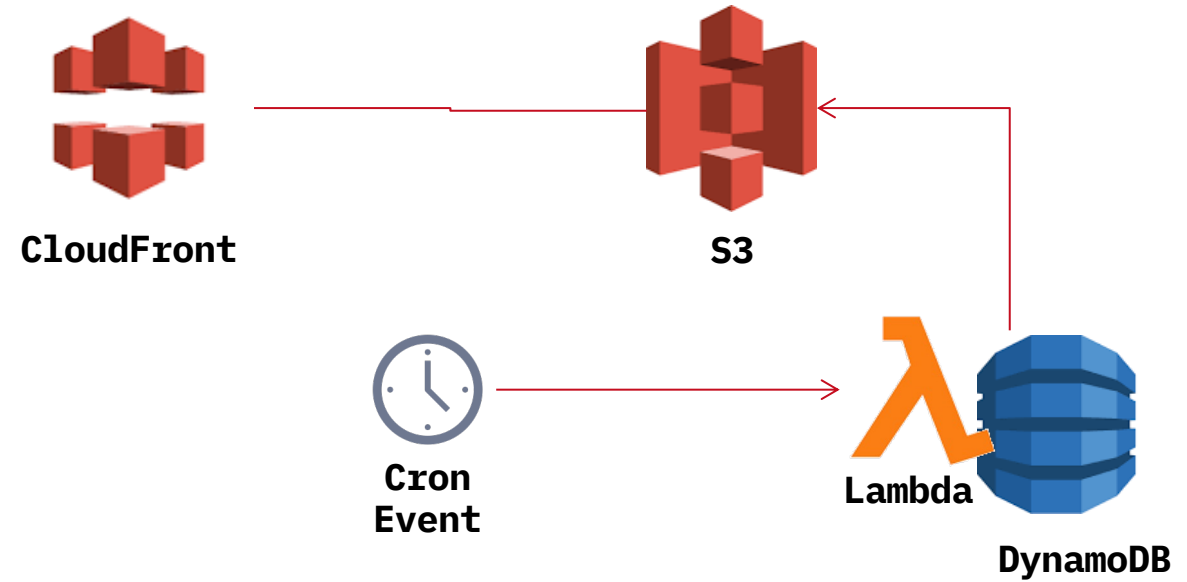
Web-Applications

- webapps with very limited interactive functionality
- webapps with very few users, e.g., CV pages
- low-cost CMS, use a function to generate new pages and serve everything statically



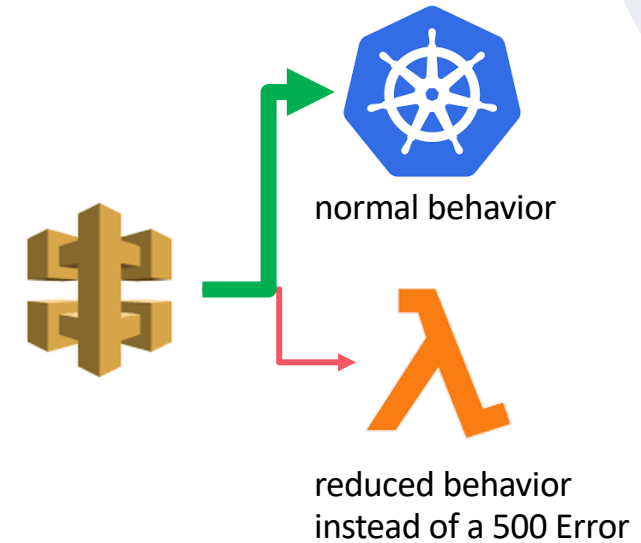
Web-Applications

- webapps with very limited interactive functionality
- webapps with very few users, e.g., CV pages
- low-cost CMS, use a function to generate new pages and serve everything statically
- Semi-static pages, e.g., content that changes slowly



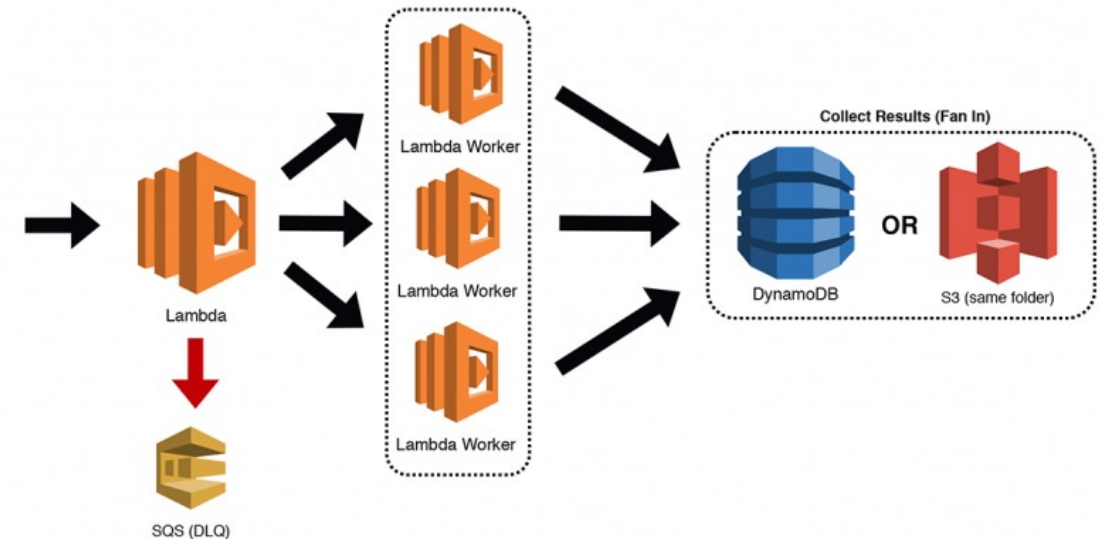
Failover Handlers

- Circuit-breaker or overflow logic, e.g., serve alternative logic in edge cases
- transaction logging, e.g., write failed messages to a dead-letter-queue
- can be costly (configuration, handling, execution) for rare events



Processing Jobs

- utilize the vast scalability of serverless platforms, on AWS you can get ~ 3000x[6vCPUx10GB] “servers” in 30s in theory
- highly flexible scalability → varying workloads can be managed by faas function
- so far only sensible for CPU/IO bound tasks

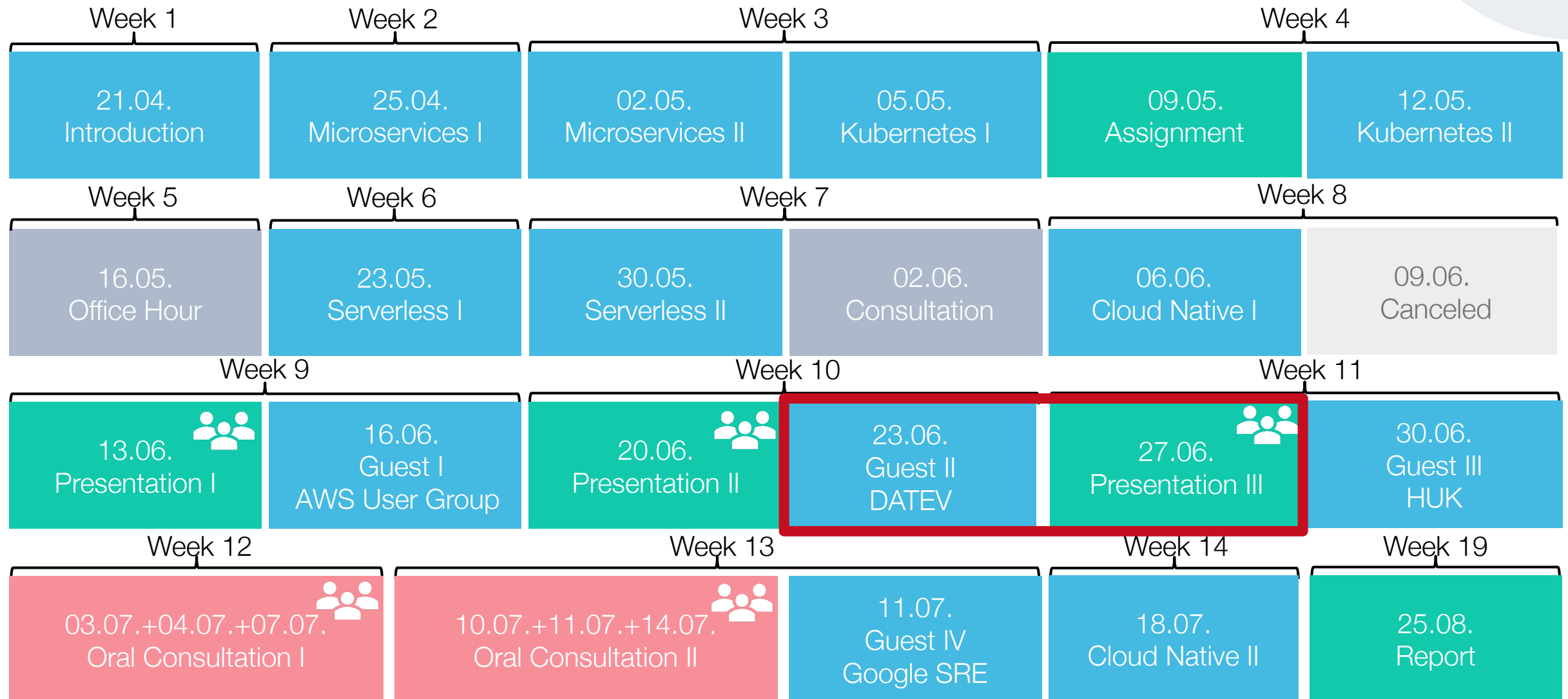


Job Interview Question



For what use cases are serverless applications not well suited?

Schedule



Reading Assignment and Presentation

Expected Artifacts **Presentation:**

- 10 min presentation
- 10 min discussion
- Each team must upload the slides as a PDF to ISIS
- **DEADLINE: 12/06/2023, 10 AM**

<https://isis.tu-berlin.de/mod/assign/view.php?id=1603410>

Design Document

Expected Artifacts:

- 4-6 Pages incl. references
- Format: IEEE Double Column
- 25%-50% generative elements
- Appendix: “creation log” – prompts you used, output you got and where you used the output in the text
- Append a brief description of who contributed what to the document
- **DEADLINE: 12/06/2023, 23:59 PM** upload via ISIS

<https://isis.tu-berlin.de/mod/assign/view.php?id=1603405>

Group Consultations 2.6.2023

- each group gets an assigned timeslot this week
- on 2.6.2023 each group has a 1 on 1 session with a topic advisor
- prepare to talk about your approach, we use this time to ensure that your presentation is a success